A VERIFIABLE ENVIRONMENT FOR AGENTS THAT DESIGN NEURAL ARCHITECTURES

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning and agent evaluation both need a *verifier*: a function that scores a candidate cheaply, with no human in the loop. While software development relies on compilers and mathematics on automated answer checkers, neural architecture design has lacked a deterministic, public verifier. We present an environment in which an agent designs a neural network by editing a typed graph, and a deterministic verifier grades the result in under a millisecond against structural, budget, and serving-cost constraints. No human and no LLM takes part in scoring. The environment includes a procedural task generator (ten families, with satisfiability and non-vacuity checks and red-teamed anti-gaming rules), a difficulty-calibration harness, and a grounding study. In that study, across 264 architectures, the verifier blocks every corrupted graph ($184/184$), no blocked graph survives a real PyTorch forward pass ($0/184$), and every clean graph runs ($80/80$). The 0 to 100 health score is a validity margin, partly a size proxy, and not a quality ranking. We report four findings. *As a repair signal*, verifier feedback lifts `claude-sonnet-4-6` from 67% to 93%, `grok-4` from 78% to 96%, and `deepseek-chat` from 48% to 93%; the lift concentrates on arithmetic defects, while cross-branch topology is largely unfixed. *As a correction to how design agents are scored*, a rubric that does not propagate shapes rates the same models 13 and 14 points higher than they are. *As a training-data generator*, fine-tuning a 1.5B model on 3,000 verified pairs lifts pass@1 from 17% to 76% on held-out tasks from the same ten families, and 100 GRPO steps on top reach 80% (n=192). Under a stricter rubric the same recipe runs $15\% \rightarrow 70\% \rightarrow 80\%$, and a 7B policy starts lower than the 1.5B (5%, because it is more verbose) and ends at 76%. Transfer is not general: four families held out of training collapse to 0%. A contamination audit we ran on ourselves shows the naive protocol would have reported 86%. *As ground truth*, eleven LLM reward models have a 0% false-positive rate on blatantly broken designs (ten of eleven when re-measured), yet approve 24 to 48% of near-miss designs that are one edit from correct. The verifier is 0% on both tiers. The same function serves as a leaderboard, an RL gym, and a verified-data generator. We'll open-source the full environment, verifier, calibration suite, and fine-tuning datasets.

Reproducibility note: Every number in this paper is reproduced by a command in the repository. The calibration, amplification, and reward-model results need the provider API keys and GPUs; the rest need neither. `VERIFICATION.md` in the repository maps each headline number to its source artifact or reproduction command. Code and data: anonymized repository in the supplementary material; released under MIT upon publication.

1 INTRODUCTION

The scarce ingredient in agent training and evaluation is the verifier, not the policy. SWE-Bench [1] grades a patch with the project's own tests, τ-bench [2] grades a dialogue by diffing database state, and mathematics has programmatic answer checking [14]. None of them needs a human judge or a second model. That is why they are trusted, and why they can be training signals.

Neural architecture design has lacked such a verifier, for a structural reason. Whether a proposed network can run at all (head counts that divide the embedding dimension, linear widths that match

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

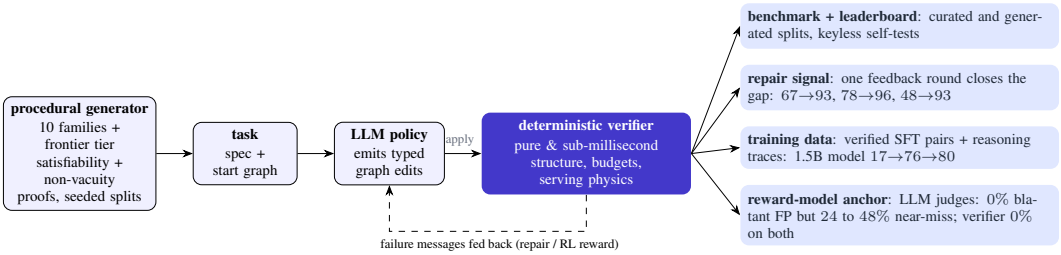


Figure 1: One pure function, four uses. A procedural generator (satisfiability-checked, seed-addressable) produces tasks. An LLM policy edits a typed graph. A deterministic sub-millisecond verifier grades structure, budgets, and serving cost. The same verdict is the benchmark score, the repair signal fed back to the policy (and the RL reward), the filter that produces verified training data, and the ground truth that audits LLM reward models.

upstream shapes, an input that reaches the output, a parameter count inside a budget) is a pure function only when the architecture is a typed graph rather than free-form code. Coding agents emit code, not graphs, so their output cannot be checked at the shape level. We close this gap by building the environment around a typed graph and the deterministic checks that a production editor already runs on it. Existing structured views, such as Netron [13], are read-only: they render a graph but do not edit it, propagate shapes, or verify, so they cannot serve as an action space or a reward.

Our contributions are: (1) a design-from-spec and edit-in-place environment with a sub-millisecond programmatic verifier; (2) a procedural task generator with satisfiability checks, non-vacuity checks, and anti-gaming constraints; (3) a calibration harness that measures difficulty against the band labs consider usable, with keyless self-tests that check the harness itself; (4) a grounding study that connects verdicts to real PyTorch behavior, with its negative results included; and (5) the empirical results in the abstract, released with the environment, together with the contamination audit that qualifies them. Figure 1 summarizes the system.

2 The Environment

State. A typed graph: components from a vocabulary of 182 layer types, each carrying parameters, connected by directed edges. **Action.** A batch of structured edits (`add_component`, `add_connection`, `update_params`, `delete_component`, `replace_model`, and others), the same vocabulary a production editor executes. **Reward.** A 0 to 100 health score plus a list of hard blockers. Figure 7 shows the full loop.

The environment is a deterministic single-agent MDP. Its state is a typed graph, its actions are batches of typed edits, and its reward is the grader's health score under a success predicate over the task's constraints. Transition and reward are pure, so an episode is reproducible from its seed (Appendix A).

A task pairs a natural-language specification with a starting graph and machine-checkable constraints. For example: *"This encoder fails validation: embedDim (192) is not divisible by numHeads (7). Repair the attention configuration in place with at most 2 actions. Do not rebuild the model from scratch."* Figure 8 shows this state and the blocker the verifier returns.

3 Task Generation

Curated tasks are a seed. A deterministic generator produces splits of any size from a seed across ten families: six design-from-spec (dense classifier, autoencoder, convolutional classifier, transformer encoder, grouped-query attention encoder, two-tower retrieval) and four edit-in-place (repair a broken attention configuration, trim an oversized model under a parameter budget, grow an undersized model into a two-sided parameter band, insert normalization). Tasks are synthesized rather than scraped, so a held-out split never has to appear on the public web. Train/eval separation *within* the environment is enforced by an explicit holdout exclusion, added after our audit (Section 8). A separate, opt-in *frontier tier* adds three families shaped like 2026 production models: Mixture-of-Experts blocks with router-validity blockers, long-context encoders under KV-cache byte budgets that full

Algorithm 1 GRADE(T, G): the deterministic verifier

Require: task T with constraints ϕ and threshold τ; typed graph $G = (V, E)$
Ensure: health score $h \in [0, 100]$; hard-blocker set B

$B \leftarrow \emptyset$
if $\neg$ REACHES(in(G), out(G)) **then** $B \leftarrow B \cup \{\text{DISCONNECTED}\}$
end if
for each attention node $v \in V$ **do**
  if v.embedDim mod v.numHeads $\neq 0$ **then** $B \leftarrow B \cup \{\text{HEADDIV}(v)\}$
  end if
  if v.numHeads mod v.numKVHeads $\neq 0$ **then** $B \leftarrow B \cup \{\text{GQADIV}(v)\}$
  end if
end for
for each node v with typed input width $w_{\text{in}}(v)$ and parent u **do**
  if $w_{\text{in}}(v) \neq w_{\text{out}}(u)$ **then** $B \leftarrow B \cup \{\text{SHAPEMISMATCH}(v)\}$
  end if
end for
compute PARAMS(G), KV(G), LATENCY(G); add a blocker for each budget in ϕ violated, and for each required type absent
$h \leftarrow 0$ **if** $B \neq \emptyset$ **else** $100 - \sum_{s \in \text{SOFT}(G)} \text{pen}(s)$
return (h, B)

attention cannot meet, and in-place grouped-query-attention retrofits. The same KV formula (Eq. 4) grades them. An opt-in frontier tier (MoE, KV-cache budgets, GQA retrofits) is documented in the repository; no number in this paper depends on it.

Satisfiability and non-vacuity. Every generated task carries its own reference solution. A 500-case sweep checks that each reference passes its own task and that every edit-in-place start graph fails its own task untouched. Both run in CI without keys.

Contamination. Earlier drafts called contamination resistance a property of construction. That is an argument, not a number, so we measured it. For each of 200 generated specs from the held-out seed, we ask an n-gram index over a web-scale pretraining corpus (infini-gram [18] over RedPajama) how many documents contain the spec's central 15-, 12-, 8- and 5-word spans verbatim. **None of the 200 specs has any span of five or more words anywhere in the corpus.** The control gives that number meaning. The same probe run on sentences from *Attention is All You Need* and *Deep Residual Learning* matches at 12 words in 15 to 32 documents each, so the procedure does detect overlap where it exists. The script aborts when its controls fail instead of reporting a clean benchmark. It did so twice under rate limiting during this measurement, so a throttled run cannot pass as a clean one.

Anti-gaming. Edit-in-place families forbid `replace_model` and `clear_canvas`, budgets are two-sided bands rather than ceilings, and a KV-cache budget is always paired with a required-attention constraint, so the three obvious ways to pass without diagnosing anything fail. Nine such strategies and the tests that pin them are in Appendix A.

Curated tasks. On twelve hand-authored tasks from real production architectures, `claude-sonnet-4-6` passes $8/12$ first try (a rerun landed at $7/12$; both predate the vacuity fix and include one free point). The failures are machine-checkable and ordinary: a missing required layer, a disconnected graph, a budget exceeded by an order of magnitude (Appendix A).

4 DIFFICULTY CALIBRATION

An environment that cannot be gamed is still useless at 0% pass (no learning signal) or near 100% (nothing to learn). Practitioners cite a floor near 2 to 3% pass rate on the hard end (roughly one success per 33 to 50 rollouts) for a usable RL environment. The calibration harness measures per-family

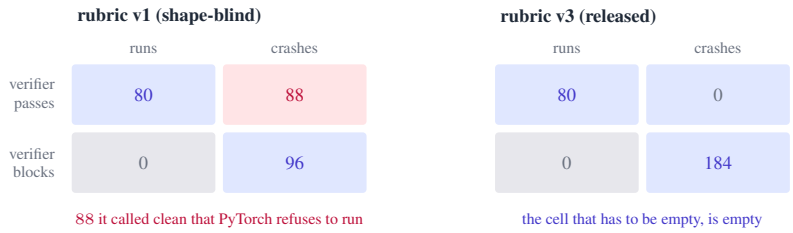


Figure 2: Verdict against reality on all 264 graphs (80 clean references, 184 systematically corrupted). Each matrix crosses what the rubric says with what PyTorch does. The cell that matters is *passes* × *crashes*: a graph the rubric certifies and the framework refuses to run. Rubric v1 puts 88 graphs there (all 80 width-corrupted ones and 8 severed ones), a 47.8% miss rate. The released rubric puts none there, and neither rubric blocks a clean graph. Of the clean graphs that run, 90% also make training progress.

pass rates with Wilson 95% intervals [15] and flags each family's band. Two keyless self-test policies check the harness itself: a `reference` policy that replays known-good solutions (measured: 100% pass) and a `noop` policy that submits empty plans (measured: 0% pass). If either self-test deviates, the harness rather than the model is broken, and CI fails.

Per-family pass rates (`claude-sonnet-4-6`, n=12/family). Five families are saturated and three sit at 0% under the released rubric, which is the spread a usable training signal needs. The three are not hard in the same way, and the harness says so (Appendix B).

5 Grounding Study

We generated clean reference architectures and systematically corrupted variants (broken attention divisibility, mismatched linear width, a severed mid-graph edge), built every graph as a PyTorch module, and trained each briefly on a synthetic fixed-teacher task.

Result on 264 graphs. The verdict agrees with physical reality on every graph (Figure 2). No blocked graph runs (0 of 184). All 184 corrupted graphs carry a blocker. Every clean graph constructs and runs (80/80), and 90% make training progress. Width and connectivity defects instantiate and only then fail the forward pass, which is why a rubric that does not propagate shapes cannot see them.

Seven corruption families. We widened the grounding study from three corruption families to seven, adding four more ways of breaking a graph and building every instance (n=12 each). Three go unblocked, which looks like three new blind spots until the graphs are run. Only one is: all 12 wrong-axis normalization graphs pass and *none* completes a forward pass, while the oversized-kernel and undersized-embedding families run (Appendix C).

Structural Deficiencies of Shape-Blind Rubrics Rubric v1 had neither the width check of Algorithm 1 line 12 nor the orphan check, and it passed 88 of 184 corrupted graphs as clean, every one a graph that cannot run (Figure 2). A two-tower graph whose user branch is cut still has an input reaching an output, so reachability alone reports it healthy. Both checks ship (Appendix C). A third blocker class, a non-merging node fed parents of differing widths, is ill-formedness rather than a proven crash, and we report it separately throughout (Appendix C).

The health score's *magnitude* does not rank training quality (Spearman $\rho = -0.02$, undefined on families whose instances all score alike). It is a validity margin, not a quality ranking, and a third of its variance is model size (Appendix C).

A learned head ranks what the score cannot. The environment produces the data for its own successor: 200 clean graphs trained to convergence give (architecture, verdict, curve) triples at no labelling cost. A ridge on structural features, scored *leave-one-family-out*, raises the rank correlation with real training behavior by half where both are defined ($|\rho| = 0.42$ against the score's 0.28) and ranks the 4 of 10 families the score scores alike (Appendix C).

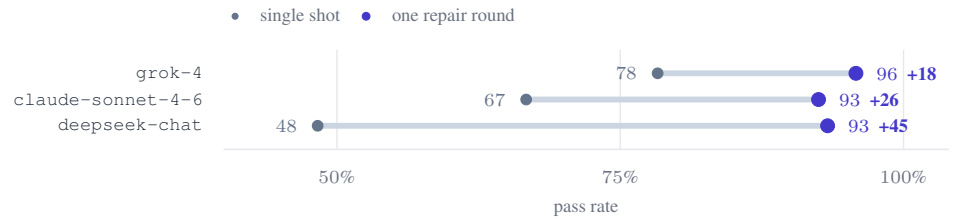


Figure 3: Verifier-in-the-loop lift for `claude-sonnet-4-6` under rubric v2. The overall gain (67 → 93; 31 tasks rescued) concentrates on the families the model finds hard: insert-norm (0 → 100, 12 fixes), the transformer encoder (8 → 100, 11), and grouped-query attention (67 → 100, 4). Two-tower retrieval is the family feedback *cannot* rescue (0 → 25): its errors are cross-branch shape mismatches that one repair round does not close. The remaining families sit at or near 100% single-shot.

6 Verifier-in-the-loop amplification

Because grading is free, the verifier's failure messages can be fed back for repair rounds. We measure, per provider, a single shot against up to k=3 rounds, holding tasks, model and prompt fixed so the only variable is the feedback loop (Algorithm 2). The framing favors the model on purpose: the environment's value is the lift it adds to any model.

Model	single-shot pass	with verifier feedback	lift
claude-sonnet-4-6	67%	**93%**	+26 pts
grok-4	78%	**96%**	+18 pts
deepseek-chat (V3)	48%	**93%**	+45 pts

Table 1: Verifier-in-the-loop lift, per provider under rubric v2 (claude and grok n=120, deepseek n=60; zero exclusions). Same tasks, model, and prompt. The only variable is the feedback loop.

The lift is concentrated on the families a frontier model finds hard (Figure 3). Of the 31 tasks feedback rescues for claude, insert-norm (0% → 100%) accounts for 12, the transformer encoder (8% → 100%) for 11, GQA (67% → 100%) for 4, two-tower for 3, and the convolutional classifier for 1. The effect is not specific to one model, and it grows as the model weakens: grok-4 goes 78.3% → 95.8% (+17.5 points, n=120) and deepseek-chat 48.3% → 93.3% (+45.0 points, n=60), both with zero exclusions. Each of these lifts is *larger* than the same measurement under rubric v1 (+21, +14, +31). That is the expected direction: a verifier that sees more defects has more to repair, and its feedback is worth more.

Re-run at HEAD under rubric v3, the claude and grok arms keep both lift and level (68% → 94% and 73% → 93%, neither outside the Wilson interval of its rubric-v2 row), so the stricter grader costs this effect nothing measurable (Appendix D).

7 Overestimation risks in shape-blind evaluation

Sections 4 and 6 report rubric-v2 numbers. The environment's earlier rubric, v1, is not a straw man. It checked attention divisibility, parameter budgets, required types, action economy, and input-to-output reachability, which is more than most hand-written graders do, and it is the rubric one writes when architectures are inspected node by node. What it could not do is follow a width along an edge. Grading the same frozen model outputs under both rubrics turns the cost of that omission into a number.

The reconstruction never favors v2, so the gaps below are lower bounds, and it reproduces the published live-v1 rates to within 0.4 points (Appendix E).

Result. A shape-blind rubric reports `claude-sonnet-4-6` 13.3 points and `grok-4` 14.3 points above their rates under a rubric that propagates shapes (Figure 4). The overstatement is concentrated. Grok's is 17 tasks, all carrying a linear layer whose declared `inFeatures` disagrees with what its parent emits, the standard bug this environment exists to catch (Appendix E).

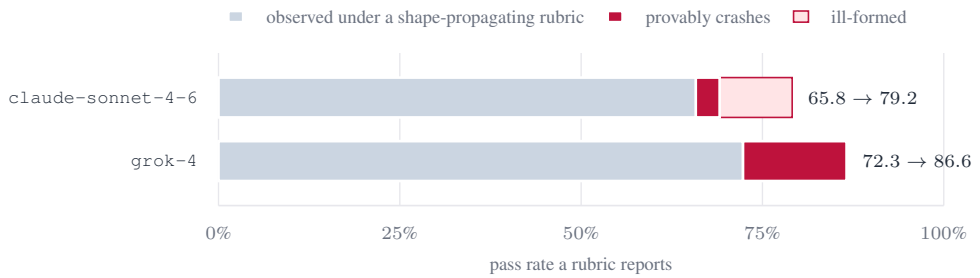


Figure 4: What a rubric that does not propagate shapes reports as competence it did not observe (generated split, seed 7, n=120; grok n=119 after one transport error). The grey span is the rate a shape-propagating rubric observes. The coloured span is what a shape-blind one adds. *Crashes*: linear widths that disagree with the upstream, and orphaned nodes; both fail a real forward pass. *Ill-formed*: a non-merging node fed parents of differing widths, where running the graph depends on edge order. The split matters. Grok's entire gap is graphs that crash; claude's is mostly two-tower graphs that are under-determined.

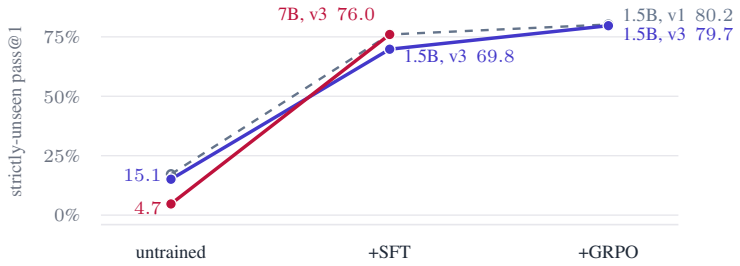


Figure 5: Two policies and two rubrics on the same strictly unseen split (n=192, seed 999; the full table is Table 2). The ordering inverts. Untrained pass@1 measures format compliance rather than design skill, so Qwen2.5-7B starts below the 1.5B (84 of 192 completions unparseable against 72) and ends above it once the environment has supplied the format. Rubric v3 propagates linear widths and grades strictly harder than the published v1 chain (dashed). It costs the 1.5B about six points and leaves the lift intact. The 7B RL stage is absent for the memory reason in Section 11.

Implications for Agent Evaluation Protocols. The verifier-in-the-loop lift says the environment helps a model. This result says something the model cannot tell you: evaluations of architecture-design agents, by default, measure something other than whether the design is buildable, and they err in the optimistic direction by roughly the size of the effects such papers report. Both numbers are reproducible from committed artifacts without an API key.

8 Training on the Environment

The environment trains models through two channels. We report both.

Verified SFT. Every generated task carries a reference solution that passes the grader, so the environment produces unlimited verified (spec → edits) pairs, without keys and for free. Fine-tuning Qwen2.5-1.5B-Instruct [17] on 3,010 of them (LoRA [16], one free T4, under ten minutes), every row verified to share no task with the evaluation split, lifts strictly held-out pass@1 by 59 points, and 100 GRPO steps add 4 more (Table 2).

A contamination audit. These are the *corrected* numbers. Our own audit found the original sampler drew from small discrete lists, so despite disjoint seeds every eval task had appeared in training. The naive protocol read 85.9% where the same-split number without overlap is 68.8%. The gap is itself a measurement, the memorization share [19], and the audit script ships (Appendix F).

Re-measured at HEAD and at 7B. Everything above was graded by rubric v1. At HEAD under v3 the recipe costs about six points of level and keeps the lift (Table 2). Scaling five-fold reverses the starting order: Qwen2.5-7B-Instruct *starts below* the 1.5B, 4.7% against 15.1%, because it is more verbose, and *ends above* at 76.0%, so untrained pass@1 measures format compliance, not design

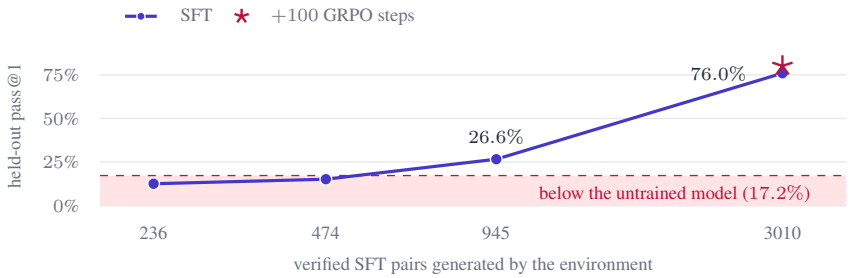


Figure 6: Held-out pass@1 against the number of verified SFT pairs (Qwen2.5-1.5B, LoRA, corrected zero-overlap protocol, seed-999 split, n=192). Below ∼1k pairs the half-learned edit format underperforms the untrained model (17.2%, dashed). 945 pairs cross above (26.6%), and 3,010 reach 76.0%. The star marks 100 GRPO steps on the 3k checkpoint (80.2%). Pairs are free to generate.

	pass@1	parse failures	mean reward
corrected protocol (v2, verified zero train/eval overlap, n=192)			
Qwen2.5-1.5B (untrained)	17.2% [13, 23]	76/192	0.19
+ SFT on 3k clean pairs	76.0% [70, 82]	14/192	1.06
+ GRPO after SFT (100 steps)	**80.2**% [74, 85]	**10/192**	**1.12**
re-measured at HEAD under rubric v3, one A100 (Section 8)			
Qwen2.5-1.5B (untrained)	15.1% [11, 21]	72/192	0.19
+ SFT on 2,824 clean pairs	69.8% [63, 76]	13/192	0.96
+ GRPO after SFT (100 steps)	**79.7**% [73, 85]	11/192	1.11
Qwen2.5-7B (untrained)	4.7% [2, 9]	84/192	−0.05
+ SFT on 2,824 clean pairs	**76.0**% [70, 82]	15/192	1.07
pre-audit protocol (v1; eval tasks seen in training)			
Qwen2.5-1.5B (untrained)	23.4% [15, 35]	19/64	0.35
+ SFT on 3k pairs	85.9% [75, 92]	2/64	1.19
+ GRPO after SFT	89.1% [79, 95]	0/64	1.26

Table 2: Training on environment-generated data (LoRA; seed-999 splits, identical eval configuration; n=192 except the pre-audit block, n=64). The first and third blocks were graded by rubric v1 on one free T4. The middle block re-runs the chain at HEAD under rubric v3 on one A100, where the stricter grader costs the 1.5B chain about six points, and the 7B policy, worse than the 1.5B untrained because it is more verbose, ends higher than the 1.5B at the same stage. Supervised fine-tuning on pairs the generator produces for free lifts pass@1 by 59 points with non-overlapping Wilson intervals. GRPO does nothing from the raw model at this scale and works after SFT. Replications, the independent reruns, and the memorization share the pre-audit block measures are reported with the training results below.

skill. Its RL stage is absent: GRPO's log-probability pass exhausts an 80GB card at the published group size. Data scaling is steep and non-monotonic: below ∼1k pairs a half-learned edit format underperforms the untrained model, 945 pairs cross above it, and 3k reach 76%. Transfer tracks data *diversity* rather than volume (Appendix F). Four families held out entirely collapse from an untrained 22.5% to **0**% while the six seen reach 78.9%: a per-family edit style, not design skill (Appendix F).

GRPO from the *raw* instruct model does nothing at this scale: the policy cannot emit parseable edits often enough for the group-relative gradient to find signal. This is the standard SFT-then-RL split, and the environment supplies both stages (Appendix F). The split recurs one level up: a 1.5B model repairs only 3% of tasks when handed the verifier's own failure text, and 204 repair-conditioned pairs plus 100 GRPO steps take that to 69.1% (Appendix F).

9 Auditing LLM reward models with the verifier

Frontier labs increasingly use a strong model *as* a reward model where the objective is not directly verifiable [12; 21]. The known failure mode is drift: the judge approves answers that are actually broken. Where a verifiable reward exists, that drift can be measured. We build a verifier-labeled set (a reference design provably passes, its unsolved starting graph provably fails) and report each

reward model	blatant FP	near-miss FP	near-miss FN
qwen-2.5-72b-instruct	0.0% [0, 2]	**48.4**% [41, 56]	0.5%
llama-3.3-70b-instruct	1.7% [0, 9]	41.5% [35, 48]	3.0%
claude-sonnet-4-6	0.0% [0, 2]	30.5% [25, 37]	5.0%
grok-4.3	0.0% [0, 6]	24.5% [19, 31]	6.5%
gpt-5.6-sol-pro	0.0% [0, 6]	23.5%	—
deterministic verifier (ours)	0%	**0**%	**0**%

Table 3: The same audit with near-miss negatives: passing designs one edit from correct (n=200 per model, 182 for qwen after provider errors; seed 11). The 0% false-positive rate on blatant negatives becomes 23.5 to 48.4%. Qwen, at 0% on blatant negatives, is the worst here and approves whatever looks plausible. LLM judges fail where reward hacking happens; the verifier does not. Rows are ordered by the near-miss rate. Sample sizes, the corruption mix and each row's provenance are below.

model	tier	agreement	false-pos.	false-neg.
llama-3.3-70b-instruct	open	98.3%	**0.0**%	1.7%
qwen-2.5-72b-instruct	open	96.7%	**0.0**%	3.3%
claude-sonnet-4-6	frontier	95.0%	**0.0**%	5.0%
grok-4.6	frontier	95.0%	**0.0**%	5.0%
grok-4.5	frontier	95.0%	**0.0**%	5.0%
grok-4.20 (reasoning)	frontier	93.3%	**0.0**%	6.7%
grok-4.3	frontier	93.3%	**0.0**%	6.7%
deepseek-r1 (reasoning)	open	93.3%	**0.0**%	6.7%
deepseek-chat (V3)	open	91.7%	**0.0**%	8.3%
mistral-large	open	88.3%	**0.0**%	11.7%
gemma-2-27b-it	open	86.7%	**0.0**%	13.3%
deterministic verifier (ours)	—	100%	**0**%	**0**%

Table 4: Eleven reward models vs. the verifier (n=60 each, generator v2). None has a false positive; the only failure mode is over-conservatism. Re-measured at HEAD under rubric v3, ten of eleven still sit at 0% and `llama-3.3-70b` moves to 1.7% (1/60). The Grok rows carry the id each response reported (Section 9). Table 3 shows this 0% does not survive near-miss negatives.

judge's **false-positive rate**: how often it approves a design the verifier proves broken. Across eleven reward models from closed-frontier (Claude, Grok) and open-weights (Qwen, Mistral, DeepSeek, Gemma, Llama) labs, the false-positive rate was 0% for every model (Table 4). The only errors were false *negatives* (1.7 to 13.3%, over-conservative), which waste valid designs but do not poison the reward. *That universal* 0% *does not survive re-measurement.* We re-ran the audit at HEAD under rubric v3, after fixing a token-budget bug that had been silently emptying completions on the OpenAI-compatible provider path (Section 11). `llama-3.3-70b-instruct` now approves 1 of 60 provably broken designs. Ten of the eleven still sit at 0%, as does a twelfth model from a lab the original audit missed (`gpt-5.6-sol-pro`, 0% over n=60; the Grok rows are re-measured with the served id verified). The qualitative claim holds and the absolute one does not: on *blatant* defects an LLM judge is nearly always safe, not always. An LLM reward model here is therefore safe but lossy. The verifier does the same job with 0% false negatives, deterministically and for free: a verdict costs microseconds of CPU, against a multi-second, paid, stochastic API call per judgment. The negatives so far, however, are blatant (an unsolved starting graph).

Vulnerability of LLM Judges to Near-Miss Reward Hacking We re-run the audit with *near-miss* negatives: passing designs corrupted by a single edit, each re-graded to confirm it fails. The same judges collapse to 23.5 to 48.4% false-positive across five labs (n=200 each, Figure 9), and Qwen, at 0% on blatant negatives, is the worst here. They are trustworthy where errors are obvious and fail where reward hacking [20] happens.

10 Related Work

Verifiable agent benchmarks. SWE-Bench [1] and τ-bench [2] established the programmatic-verifier pattern in code and tool use, and SWE-Gym [3] showed that a few thousand verifiable tasks

	domain	verifiable reward	procedural, contam.-resist.	anti-gaming red-teamed	typed-graph action space
SWE-Bench [1]	code	✓	partial	–	–
τ-bench [2]	tool use	✓	–	–	–
SWE-Gym [3]	code	✓	partial	–	–
classical NAS [11]	architecture	proxy	–	–	–
ours	**architecture**	✓	✓	✓	✓

Table 5: Where this environment sits: architecture design as a verifiable-reward domain, with a contamination-resistant procedural generator, red-teamed anti-gaming defenses, and a typed-graph action space.

suffice to train agents to a double-digit gain. We target the same pattern in a domain that lacked a public verifier (Table 5).

RL environments and verifiable rewards Training on verifiable rewards (RLVR) is now central to reasoning-model post-training [4; 5], usually with policy-gradient methods such as PPO [7] and GRPO [6], preference optimization [9], or rejection sampling in the STaR [8] style. A growing market supplies labs with RL environments. Reward-hacking resistance is the quality bar most often cited, and usable environments are expected to have a hard band near a 2 to 3% pass floor. We build against both criteria and ship GRPO and STaR loops against the same grader.

Neural architecture search Classical NAS [11], from RL-based search [22] to regularized evolution [23], automates architecture design under a proxy objective, and NAS-Bench-101 [24] made such search reproducible by tabulating a fixed cell space exhaustively. Our environment differs in two ways: it provides a human-readable, per-edit, sub-millisecond verifier over a typed graph, and it serves as a training and evaluation substrate rather than a single search procedure. Differentiable and one-shot NAS [10] make gradient search tractable by relaxing a small, hard-coded cell vocabulary and optimizing task loss; structural legality is assumed, not checked. A per-edit legality verifier is orthogonal. It can veto the candidates of *any* search procedure (a DARTS derivation, an evolutionary mutation, or an LLM agent's edits) before GPU time is spent, so it composes with these methods rather than competing with them.

Selecting among unexecuted candidates Two preference models [25; 26] and a routing formulation [27] measure the same decision. The comparison, and what our abstaining decision rule shares with them, is in Appendix H.

11 Limitations

Three limitations bound these results. *Coverage*: the verifier guarantees that its checks hold exactly, not that no defect outside them exists. Widening the grounding study from three corruption families to seven found one further blind spot. *Score semantics*: the health score is a validity margin, not a quality ranking. *Scale*: two small policies (Qwen2.5-1.5B and 7B, LoRA), the larger measured through supervised fine-tuning only because its RL stage does not fit one 80GB card at the published group size, with gains largely in-distribution. A token-budget bug on the harness's OpenAI-compatible provider path was silently emptying completions, which downstream callers scored as failed designs. It is fixed, an empty completion now raises, and every affected number was re-measured. The full register, and which measurement carries which rubric version, is in Appendix J.

12 Conclusion

Typed graphs turn "is this network sound, and what will it cost to serve" into a pure function. This paper measures what that buys: 18 to 45 points of repair lift on three frontier models, a 13 to 14 point correction to how design agents are scored, verified pairs that take a small model from 17% to 80% on strictly unseen tasks, and reward models that are near-perfect on blatant defects yet approve a quarter to a half of near-misses. We release the environment, the generator, the harnesses and the training loops.

ETHICS STATEMENT

The environment lowers the cost of designing valid, servable architectures and of training and evaluating agents that do so. This should broaden access beyond teams with dedicated ML-research staff. Two risks deserve note. First, a verifier defines what "good" means. Ours checks structural validity and serving cost, not fairness, safety, or data appropriateness, and it must not be mistaken for a quality oracle; the negative result in Section 5 is there for that reason. Second, verified-data generation can accelerate capability gains. We mitigate this by releasing the methodology openly, so the same tools that build also audit (the reward-model analysis above). The benchmark contains no personal data and no scraped content. Every task is synthetic or hand-authored.

USE OF LARGE LANGUAGE MODELS

Large language models were used in four roles, and we separate them. *As an optional task proposer*: the generator accepts an LLM-authored task only when the proposer's own reference solution passes the deterministic grader under always-enforced safety-net constraints, so no unverified LLM output enters the benchmark. *As a data source*: the two released reasoning-trace datasets consist of LLM outputs (claude, grok) kept only where the verifier passed the resulting design, as described in Appendix I. *As a writing and coding assistant*: LLMs were used, under human direction, to edit prose for readability and to draft portions of the scripts; every line of that code was reviewed and tested by the authors before any number depended on it. We have not used generative AI tools to propose or refine hypotheses, to design the methodology or experiments, to interpret results, or to formulate or prove claims; the remaining tasks in the required-disclosure list (translation, qualitative analysis, dataset cleaning) are not applicable to this work. No experimental number came from an LLM outside the measured runs: each comes from a deterministic script or a logged provider call, reproducible from the committed artifacts. We take responsibility for the final content of this work, including text, claims and artifacts produced with the aid of generative AI.

REPRODUCIBILITY

```
unzip supplementary.zip   # the supplementary material
cd arch-bench

# harness self-tests: satisfiability, non-vacuity, anti-gaming
npx vitest run
node calibrate.mjs --policy=reference   # must be 100%
node calibrate.mjs --policy=noop        # must be 0%

# mint verified pairs, holdout-excluded
node training/build_sft_dataset.mjs --count=3000 --seed=20260716

# 17.2% -> 76.0% strictly-unseen, on one free T4
python training/train_sft.py --data arch-design-sft.chat.jsonl
python training/train_grpo.py --eval-only --seed 999 --count 192 \
     --model out/sft-arch/checkpoint-final

# SFT-then-RL -> 80.2% (79.7% at HEAD, rubric v3)
python training/train_grpo.py --steps 100 --lora --lr 1e-5 \
     --model out/sft-arch/checkpoint-final

# the verifier's verdict against real PyTorch behavior
node training/dump_grounding_set.mjs --count=80 --seed=123 \
     --out=g.jsonl
python training/grounding.py --set g.jsonl --steps 60
python training/grounding_at_scale.py --set g.jsonl
python training/analyze_grounding.py
```

REFERENCES

[1] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? *ICLR*, 2024.

[2] S. Yao, N. Shinn, P. Razavi, and K. Narasimhan. τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. *arXiv:2406.12045*, 2024.

[3] J. Pan, X. Wang, G. Neubig, et al. Training Software Engineering Agents and Verifiers with SWE-Gym. *arXiv:2412.21139*, 2024.

[4] N. Lambert et al. Tulu 3: Pushing Frontiers in Open Language Model Post-Training. *arXiv:2411.15124*, 2024.

[5] DeepSeek-AI (D. Guo et al.). DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *arXiv:2501.12948*, 2025.

[6] Z. Shao et al. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. *arXiv:2402.03300*, 2024.

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms. *arXiv:1707.06347*, 2017.

[8] E. Zelikman, Y. Wu, J. Mu, and N. D. Goodman. STaR: Bootstrapping Reasoning with Reasoning. *NeurIPS*, 2022.

[9] R. Rafailov, A. Sharma, E. Mitchell, S. Ermon, C. D. Manning, and C. Finn. Direct Preference Optimization. *NeurIPS*, 2023.

[10] H. Liu, K. Simonyan, and Y. Yang. DARTS: Differentiable Architecture Search. *ICLR*, 2019.

[11] T. Elsken, J. H. Metzen, and F. Hutter. Neural Architecture Search: A Survey. *JMLR*, 20(55):1–21, 2019.

[12] L. Zheng et al. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. *NeurIPS*, 2023.

[13] L. Roeder. Netron: Visualizer for neural network, deep learning and machine learning models. Software, https://github.com/lutzroeder/netron.

[14] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, et al. Training Verifiers to Solve Math Word Problems. *arXiv:2110.14168*, 2021.

[15] E. B. Wilson. Probable Inference, the Law of Succession, and Statistical Inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927.

[16] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. LoRA: Low-Rank Adaptation of Large Language Models. *ICLR*, 2022.

[17] Qwen Team (A. Yang et al.). Qwen2.5 Technical Report. *arXiv:2412.15115*, 2024.

[18] J. Liu, S. Min, L. Zettlemoyer, Y. Choi, and H. Hajishirzi. Infini-gram: Scaling Unbounded n-gram Language Models to a Trillion Tokens. *arXiv:2401.17377*, 2024.

[19] N. Carlini, D. Ippolito, M. Jagielski, K. Lee, F. Tramèr, and C. Zhang. Quantifying Memorization Across Neural Language Models. *ICLR*, 2023.

[20] J. Skalse, N. H. R. Howe, D. Krasheninnikov, and D. Krueger. Defining and Characterizing Reward Gaming. *NeurIPS*, 2022.

[21] H. Lightman, V. Kosaraju, Y. Burda, H. Edwards, B. Baker, T. Lee, J. Leike, J. Schulman, I. Sutskever, and K. Cobbe. Let's Verify Step by Step. *ICLR*, 2024.

[22] B. Zoph and Q. V. Le. Neural Architecture Search with Reinforcement Learning. *ICLR*, 2017.

[23] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized Evolution for Image Classifier Architecture Search. *AAAI*, 2019.

[24] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter. NAS-Bench-101: Towards Reproducible Neural Architecture Search. *ICML*, 2019.

[25] Y. Zheng et al. Can We Predict Before Executing Machine Learning Agents? *arXiv:2601.05930*, 2026.

[26] C. Foster et al. AI Research Preference Models. *arXiv:2608.13940*, 2026.

[27] T. Feng, F. Yu, H. Zhang, et al. LLMRouter: Unified Infrastructure for Developing, Evaluating, and Deploying LLM Routers. *arXiv:2608.06867*, 2026.

A Environment and task generation

Widths, parameters, and cost. Each node type defines an output width w_{out} and a parameter count from its typed parameters. For the main types,

$$\text{linear}(d_{\text{in}}, d_{\text{out}}): \quad w_{\text{out}} = d_{\text{out}}, \quad |\theta| = d_{\text{in}}\, d_{\text{out}} + d_{\text{out}}, \tag{1}$$

$$\text{attention}(d, H): \quad w_{\text{out}} = d, \quad |\theta| = 4d^2 + 4d, \tag{2}$$

$$\text{conv}(c_{\text{in}}, c_{\text{out}}, k): \quad w_{\text{out}} = c_{\text{out}}, \quad |\theta| = c_{\text{in}}\, c_{\text{out}}\, k^2 + c_{\text{out}}. \tag{3}$$

Total parameters are $\textsc{Params}(G) = \sum_{v \in V} |\theta(v)|$, and the decode-time KV-cache cost per token is

$$\textsc{Kv}(G) \;=\; \sum_{v \,\in\, \text{attn}(G)} 2\, H_v^{\text{KV}}\, \frac{d_v}{H_v}\, b, \tag{4}$$

for b bytes per cached value (multi-head attention is the case $H_v^{\text{KV}} = H_v$; grouped-query attention shrinks H_v^{KV}). A task with constraints ϕ and threshold τ is solved iff

$$\textsc{Pass}(T, G) \;=\; \big[\, B(G) = \emptyset \,\big] \;\wedge\; \big[\, h(G) \geq \tau \,\big] \;\wedge\; \phi(G), \tag{5}$$

where $B(G)$ is the hard-blocker set of Algorithm 1, $h(G)$ is its health score, and $\phi(G)$ conjoins the machine-checkable constraints: required types, a two-sided parameter band $p_{\min} \leq \textsc{Params}(G) \leq p_{\max}$, KV and latency budgets, and an action-economy limit $|a| \leq a_{\max}$. The soft-penalty weights inside h are implementation detail. Every result in this paper depends only on the success predicate, and the score is diagnostic. Every term is a pure function of G, so Eq. (5) is decidable in $O(|V| + |E|)$.

The grader runs three checks, each pure and sub-millisecond: a structural score with hard blockers (disconnected input or output, attention divisibility, parameter bloat), a guardrail pass over the proposed edits, and a shape propagator that catches the shape bugs an edit would introduce. No LLM grades anything, so there is nothing to persuade and no randomness in the reward.

The checks. The grader runs three families of checks, in order. (i) *Structural blockers*: a disconnected input or output; an attention layer whose `embedDim` is not divisible by `numHeads` (or `numHeads` not divisible by `numKVHeads` under grouped-query attention); a linear layer whose `inFeatures` does not match the upstream width; a parameter count far over budget. (ii) *Serving cost*: parameters, forward multiply-accumulates, and KV-cache bytes per token, computed with the standard formula (grouped-query attention caches $2 \cdot \texttt{numKVHeads} \cdot d_{\text{head}} \cdot b$ bytes per token at b bytes per value; multi-head attention is the special case $\texttt{numKVHeads} = \texttt{numHeads}$), and checked against the task's parameter, KV, and decode-latency budgets. (iii) *Shape propagation*: a symbolic forward pass over the typed graph that catches interface mismatches before any tensor is allocated. Each pass is $O(|V| + |E|)$ and finishes in well under a millisecond. That speed is what makes the reward usable inside an RL loop rather than only as an offline evaluation.

Anti-gaming. Edit-in-place families forbid `replace_model` and `clear_canvas`, so an agent that cannot diagnose a defect cannot pass by regenerating the whole network. Budgets are two-sided bands rather than ceilings, so "delete everything" fails a trim task. A KV-cache budget is always paired with a required-attention constraint, so removing attention cannot zero the cache. Nine such strategies, their defenses, and the tests that fail if any defense is weakened are listed in the repository's red-team report (`ROBUSTNESS.md`). An LLM-driven task proposer accepts nothing without a satisfiability check: the proposer's own reference must pass the grader under always-enforced safety-net constraints. LLM authorship is unsafe for human- or LLM-judged benchmarks, and safe here.

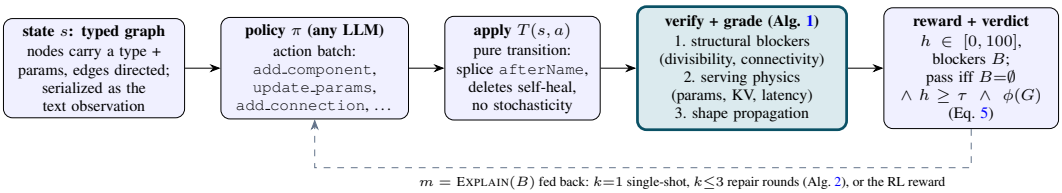


Figure 7: The environment loop in detail. The state is a typed graph, the action is a batch of structured edits, the transition is a pure function, and the grader runs three check families (Algorithm 1). The verdict is both the benchmark score and the RL reward. Feeding its failure messages back gives the repair setting of Section 6 ($k{=}1$ is single-shot). Everything after the policy is deterministic, so an episode is reproducible from its seed.

A vacuous task the check found. That sweep covers the generated families. Until rubric v3, the twelve hand-authored curated tasks were covered only by review, and review missed one. We graded every curated task with an *empty* action plan, the cheapest vacuity probe there is. One task passed. `scale-under-budget` asks for a bigger transformer under a 100M ceiling but constrains only the ceiling, so the untouched start graph (9M parameters, score 63) already met every constraint. One task in twelve could not tell any two policies apart. Rubric v3 gives the task the two-sided band it always intended (`minParams` $=$ 15M), the probe now returns $0/12$, and a test pins the fix. Curated pass rates published before v3 therefore include one free point. The generated-split numbers were always covered by the non-vacuity check and are unaffected.

Formalization. The environment is a deterministic single-agent MDP $(\mathcal{S}, \mathcal{A}, T, R)$. A state $s \in \mathcal{S}$ is a typed graph (V, E): each node in V carries a type and a parameter dictionary, and the edges E are directed. An action $a \in \mathcal{A}$ is a batch of typed edits from the fixed vocabulary. The transition $T(s, a)$ applies them, with no randomness. The reward is the grader's health score $R(s') \in [0, 100]$. A task's success predicate is $R(s') \geq \tau \wedge \neg\,\mathrm{blocked}(s') \wedge \phi(s')$, where ϕ collects the task's machine-checkable constraints (required types, parameter, KV and latency budgets, and action-economy limits). Because T and R are pure functions, an episode is fully reproducible from its seed, and the same R is at once an evaluation metric, an RL reward, and a data-generation filter.

The per-type parameter and KV-cache formulas, and the serving-cost subset the grader covers, are in Appendix A.

Curated tasks. The twelve hand-authored curated tasks come from real production architectures (a CIFAR CNN, a DLRM click-through-rate ranker, a Behavior Sequence Transformer, a two-tower retrieval encoder, a semantic-search encoder, and repair and scaling tasks). `claude-sonnet-4-6` passes $8/12$ first try (mean health score 81). An independent rerun of the same split landed at $7/12$ at a mean of 70, so read it as 7 to 8 of 12. Both runs predate the vacuity fix above and include one free point. Appendix K sketches four of the tasks as typed graphs. The failures are machine-checkable and ordinary (Table 9): the model omits the required attention layer or exceeds the parameter budget by an order of magnitude. A benchmark a frontier model clears $12/12$ measures nothing.

Satisfiability and non-vacuity. Every generated task carries its own reference solution. A 500-case sweep checks that each reference passes its own task (no unsatisfiable tasks) and that every edit-in-place start graph fails its own task untouched (no vacuous tasks). Both run in CI without keys.

The same probe found one of the twelve curated tasks that an *empty* action plan passed, because it constrained only a ceiling its start graph already met. Rubric v3 gives it the two-sided band it intended, so curated rates published before v3 carry one free point (Appendix A).

B DIFFICULTY CALIBRATION, IN FULL

Per-family pass rates (`claude-sonnet-4-6`, $n{=}12$/family). Five families are saturated and three sit at or near 0% (Table 6). The three are not hard in the same way, and the harness says so. Transformer-encoder failures omit a required layer and two-tower failures are structural. Every

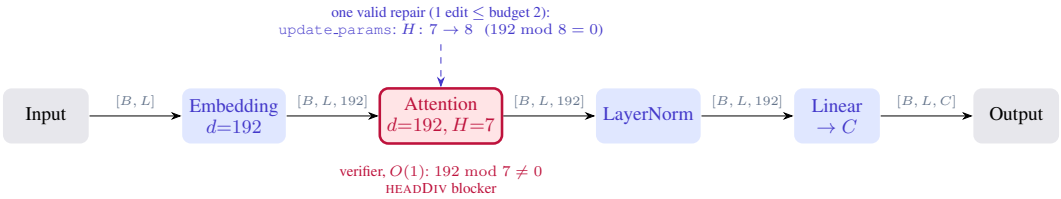


Figure 8: A task state as a typed graph. The one node the verifier blocks is red and every other node is neutral. Each edge carries the symbolic shape the verifier propagates (B batch, L sequence length). The verifier returns the hard blocker at the attention node in $O(1)$ ($192 \bmod 7 \neq 0$; Algorithm 1, line 5). One valid repair is a single `update_params` inside the task's 2-edit budget. Widths and divisibility are properties of typed nodes, so every annotation here is a pure function of the graph. On free-form code each would require executing the model.

Family	single-shot	with verifier feedback
dense classifier	100%	100%
autoencoder	100%	100%
repair attention	100%	100%
trim under budget	100%	100%
grow to band	100%	100%
convolutional classifier	92%	100%
GQA encoder	67%	100%
transformer encoder	**8%**	**100%**
insert normalization	**0%**	**100%**
two-tower retrieval	**0%**	**25%**
overall	**67%**	**93%**

Table 6: Per-family pass rate for `claude-sonnet-4-6` under rubric v2 (generator v2, seed 7; n=12 per family, all 120 tasks graded with zero exclusions). Five families are saturated. Feedback closes the transformer-encoder and insert-norm gaps completely. Two-tower retrieval is the family feedback cannot rescue, and it is also the family rubric v1 rated at 100% (Section 7). Overall single-shot 66.7% (Wilson 95% CI [58, 74]) rises to 92.5% ([86, 96]).

insert-norm failure, by contrast, is a graph the verifier scores 94 to 100 that spends three to five times the task's action budget: the model can design it but not economically. Reporting an action-economy limit and a design failure in one band would flatter the difficulty curve, so we label it.

The frontier tier is hard for one model only. An independent calibration of the three frontier families (seed 11, n=12/family) puts `claude-sonnet-4-6` at 27.8% overall, inside the useful band on all three families (25%, 25%, 33%), and `grok-4` at 97.2%, saturated on all three (92%, 100%, 100%). A tier built to be uniformly harder is, for one of the two frontier models we tested, easier than the core tier. We report both columns because difficulty here is a property of the model-task pair, not of the task family. An environment sold on having a hard band should say which model it was hard for.

C GROUNDING STUDY, IN FULL

Negative result. Within clean graphs, the magnitude of the 0 to 100 score does *not* rank training quality. We trained all 80 clean architectures to convergence (800 steps, fixed-teacher task distribution) and correlated the score against best achievable loss. Across the corpus, Spearman $\rho = -0.02$: no relationship. The cross-corpus number is confounded by task difficulty, so we also report it within families. On the four families whose eight instances all receive the *same* score, ρ is undefined: the score does not discriminate there at all. On the families where it does vary it ranks designs in the right order, sometimes strongly (trim $+0.85$, transformer encoder $+0.74$, dense classifier $+0.43$). Against this, $\rho(\text{score}, \text{parameters}) = -0.56$, so roughly a third of the score's variance is model size rather than anything about the design. The score is therefore a validity margin, not a quality ranking, and its magnitude is partly a size proxy. The natural next step is a learned quality head trained on

(architecture, verdict, real training curve) triples, which the environment can produce at scale. We validate this predictive formulation in the following experiment.

A learned head ranks what the score cannot. The environment produces the training data for its own successor. 200 clean graphs, each built in PyTorch and trained to convergence, give (architecture, verdict, curve) triples at the cost of GPU time and no labelling. We fit a ridge regression on structural features (parameter count, depth, width statistics, the fraction of each layer class, and the health score itself) and score it *leave-one-family-out*, with family identity absent from the features, so the head cannot win by learning that convolutional graphs converge slowly. The target is the area under the loss curve rather than final loss, because the probe task is easy enough that a third of clean graphs reach exactly zero and final loss barely varies.

On the families where both predictors are defined, the head raises the rank correlation with real training behavior by half: mean $|\rho| = 0.42$ against the health score's 0.28. The more important number is where the score is silent. On 4 of 10 held-out families every instance receives the *same* score, so its rank correlation is undefined; the head ranks those families too (mean $|\rho| = 0.42$ over all ten). The heaviest features are the convolutional fraction and the width dynamic range. The health score's own weight is -0.07, near zero, so the score contributes almost nothing to predicting how a design will train. That is the same negative result as above, now stated as a coefficient. This does not make the verifier a quality oracle, and one ridge on 200 graphs is a small result. It does show that the gate and the predictor are separable, and that the environment supplies the data for the second at no labelling cost.

Seven corruption families. The study above uses three ways of breaking a graph, so it bounds the verifier's blind spot only over those three. We added four more, each a defect practitioners actually ship: a normalization layer sized to the wrong axis, a convolution whose channel count disagrees with its predecessor, a kernel wider than the feature map it slides over, and an embedding table too small for its own indices. The verifier blocks every instance of the convolution defect and none of the other three. That looks like three new blind spots. Running the graphs says otherwise (n=12 per family, same fixed-teacher probe). *Normalization width is a real blind spot*: all 12 pass the verifier and *none* completes a forward pass (`running_mean should contain 253 elements not 266`). The other two are not defects the verifier should catch: all 12 oversized-kernel graphs run, and all 12 undersized-embedding graphs both run and train, as does the clean control ($12/12$ run, $10/12$ train). A block-rate table alone therefore overstates a verifier's blind spot, because it counts unblocked graphs without asking whether they were broken. Only executing them separates the two. The precision claim is unchanged (no blocked graph runs), and the recall gap is now measured over seven families rather than asserted over three.

Structural Blind Spots of Shape-Agnostic Rubrics. Rubric v1, released with an earlier draft, had neither the width check of Algorithm 1 line 12 nor the orphan check. Under v1, all 80 width-corrupted graphs and 8 of the 80 severed graphs passed as clean: an $88/184$ (47.8%) miss rate, and every one of them is a graph that cannot run. The severed cases are the instructive ones. A two-tower graph whose user branch is cut still has an input reaching an output, so a reachability test alone reports it healthy, and only per-node in-degree exposes the dead branch. v1 also certified one of our own twelve curated reference solutions, which widened a transformer's blocks while leaving the embedding and the classifier head at the original width. The oracle it blessed was a model PyTorch refuses to build. Both checks now ship in the released grader, pinned by regression tests. The current released rubric is v3, which adds the `minParams` floor that fixes `scale-under-budget` (Section 3). Section 7 measures what the same blindness does to a live model's score, and Section 11 records which measurements still carry v1 numbers.

Result on 264 graphs. The 264 graphs are 80 clean references and 184 corrupted variants in three families: 24 with attention divisibility broken, 80 with a mid-chain linear width doubled, and 80 with an interior edge severed. The verdict agrees with physical reality on every graph (Table 7, Figure 2). *Precision*: no blocked graph runs; 0 of 184 complete a forward pass. *Recall*: no corrupted graph escapes; all 184 carry a blocker. *Specificity*: every clean graph constructs and runs ($80/80$), and 90% make training progress. The failure modes separate by family. Broken divisibility fails at construction (none of the 24 instantiate). Width and connectivity defects instantiate and then fail the forward pass. That is why a rubric that inspects nodes without propagating shapes cannot see them.

class	n	blocked	constructs	forward pass	trains
clean reference	80	0	80	**80**	72
divisibility broken	24	24	0	**0**	0
linear width doubled	80	80	80	**0**	0
interior edge severed	80	80	80	**0**	0
all corrupted	184	**184**	160	**0**	0

Table 7: Verifier verdict against real PyTorch behavior, per corruption family (rubric v2, seed 123, 60-step probe). Verdict and physics agree on all 264 graphs: nothing blocked runs, nothing corrupted escapes, and every clean graph runs. Reproduce with `dump_grounding_set.mjs` followed by `grounding.py`; neither an API key nor a GPU is required.

One blocker class is ill-formedness, not a crash. The width and orphan checks predict runtime failure. A third check does not, and we separate it rather than count it. When a non-merging node receives parents of differing widths, the graph has no single defined behavior. Our own reference builder resolved it by taking the first parent and discarding the others, so such graphs *ran*. That resolution is arbitrary, and we can show it. Take a two-tower graph `claude-sonnet-4-6` actually produced, whose `score_head` receives both a 664-wide concatenation and a 332-wide tower. Swapping only the order of those two incoming edges flips the graph from training normally to `mat1 and mat2 shapes cannot be multiplied (64x332 and 664x1)`. Whether it runs depends on the order its edges were emitted in, not on the design. We therefore treat such graphs as ill-formed rather than as proven crashes, we fixed the builder to raise instead of silently choosing, and we report determinacy defects separately from shape defects throughout (Figure 4).

D VERIFIER-IN-THE-LOOP AMPLIFICATION, IN FULL

Topological Failure Modes Invariant to Verifier Feedback. Three rounds do not close the gap, and the residue is almost entirely one family. Every surviving claude failure is a two-tower graph, and every one carries the same blocker: a node reached by two branches of different widths. These are the graphs of Section 5's determinacy class, and the models produce them in a specific way. Given a spec that says *both tower outputs must have identical shape for the merge*, the model builds the concatenation and the scoring head correctly and then leaves an extra edge from one tower straight to the output, so the output is both a 429-wide vector and a scalar score. Feedback naming the defect three times over does not get it removed. This is the one place where the environment's central claim does not hold: verifier feedback repairs arithmetic reliably and topology poorly.

The verifier as an in-reasoning tool. Repair rounds consume feedback *after* a submission. Reasoning models are also trained to call tools *during* reasoning. We expose the verifier as a function tool (`audit_architecture` for a dry-run grade, `submit_actions` to finish) and compare the same model with and without it on identical tasks. grok-4 rises from 85.7% ($24/28$ graded) to 100% ($29/29$) on the standard split at a mean of 1.17 audit calls per task, and from 96.6% to 100% on the frontier tier (n=30 per tier; provider errors excluded per arm, two and one on the generated split, one and none on the frontier). The tool interface also removes free-text parse failures, because function calls are structured. This is the tool-integrated-reasoning setting frontier labs train in, with a deterministic tool. The released harness includes a keyless self-check for it.

The lift survives a stricter grader. We re-ran the claude and grok arms at HEAD under rubric v3, on the same tasks and seeds. Both the lift and the levels stay where they were: claude $68\% \to 94\%$ ($82/120 \to 113/120$, a $+26$ lift against $+26$ under v2), grok $73\% \to 93\%$ ($88/120 \to 112/120$, $+20$ against $+18$), every rescue again at the second or third turn, and no provider errors. Neither level moves outside the Wilson interval of a 120-task split, so the stricter rubric costs this effect nothing we can measure. The grok row also confounds rubric with model, because xAI now serves the `grok-4` id as grok-4.3. `deepseek-chat` was not re-run for want of provider access, so its rubric-v2 row stands alone.

Algorithm 2 ROLLOUT(T, π, k): verifier-in-the-loop repair

Require: task T; policy π; maximum repair rounds k
Ensure: pass $\in \{0, 1\}$; rounds used
$G \leftarrow T$.startGraph; $\quad m \leftarrow \varnothing$ ▷ m: verifier feedback
for $t \leftarrow 1$ **to** k **do**
  $a \leftarrow \pi\big(T.\text{spec},\ \text{SERIALIZE}(G),\ m\big)$ ▷ batch of typed edits
  $G \leftarrow \text{APPLY}(G, a)$
  $(h, B) \leftarrow \text{GRADE}(T, G)$
  if $B = \emptyset \ \wedge\ h \geq \tau \ \wedge\ \phi(G)$ **then return** $(1, t)$
  end if
  $m \leftarrow \text{EXPLAIN}(B)$ ▷ the verifier's failure messages
end for
return $(0, k)$

max repair rounds k	claude-sonnet-4-6	grok-4	deepseek-chat
$k = 1$ (single-shot)	66.7% [58, 74]	78.3% [70, 85]	48.3% [36, 61]
$k = 2$	90.8% [84, 95]	90.0% [83, 94]	90.0% [80, 95]
$k = 3$	**92.5**% [86, 96]	**95.8**% [91, 98]	**93.3**% [84, 97]

Table 8: Ablation on the number of verifier-feedback repair rounds under rubric v2, three models (generator v2, seed 7; claude and grok n=120, deepseek n=60, zero exclusions; Wilson intervals). One round carries most of the lift but not all of it: a second round still adds 1.7, 5.8, and 3.3 points. Under rubric v1 the same ablation showed k=3 adding nothing, because the defects a third round fixes are shape defects v1 could not see.

E QUANTIFYING PERFORMANCE OVERESTIMATION IN SHAPE-BLIND EVALUATION

Result. A shape-blind rubric reports `claude-sonnet-4-6` 13.3 points and `grok-4` 14.3 points above their rubric-v2 rates (Figure 4). The overstatement is concentrated. For grok it is 17 tasks, all carrying a linear layer whose declared `inFeatures` disagrees with what its parent emits, and the mistake has one shape: the model computes a flattened width (`head` declaring 73,984, 81,152, 129,536) where the upstream emits 256. That is the standard bug this environment exists to catch, produced by a frontier model on more than one task in seven, and invisible to a rubric one edge short of seeing it. For claude it is 16 tasks: 12 two-tower graphs in the ill-formed class and 4 outright crashes. On the curated split the gap is 0.0 for both models, because there every width defect arrives with an independently visible one. Twelve hand-authored tasks are too few to surface the class, which is itself an argument for a generator.

Method. A leaderboard artifact records, per task, the verdict and every failure string. A task is scored as *v1 would have passed* when each of its failures is either a v2-only blocker or the low-score line that blocker mechanically produces, because GRADE zeroes any blocked graph's score. A task with an independent failure (over budget, missing type, too few components, action limit) failed under both rubrics and is not flipped. The reconstruction therefore never favors v2, and the gaps below are lower bounds. It is also checkable: on the generated split it reports 79.2% for `claude-sonnet-4-6` and 86.6% for `grok-4`, against 79.2% and 86.2% published from live v1 runs. Reproducing those to within 0.4 points justifies reconstructing rather than paying to re-run a rubric we no longer ship (`rubric_delta.mjs`, keyless).

F TRAINING ON THE ENVIRONMENT, IN FULL

Verified SFT. Every generated task carries a reference solution that passes the grader, so the environment can produce unlimited verified (spec → edits) supervised pairs, without keys and for free. We fine-tuned Qwen2.5-1.5B-Instruct [17] on 3,010 such pairs (LoRA [16], one free T4, under ten minutes), with every training row verified to share no task with the evaluation split. Strictly held-out pass@1 rises from 17.2% (33/192, Wilson [13, 23]) to 76.0% (146/192, [70, 82]), and 100 GRPO steps on the SFT checkpoint reach **80.2**% (154/192, [74, 85]; Table 2, Figure 5). An independent

rerun of the GRPO stage from the same SFT checkpoint reproduces the headline exactly (154/192). The secondary metrics move within run-to-run noise (parse failures 10 vs. 14, mean reward 1.12 vs. 1.10), so we treat the pass@1 gain, and not the parse-failure reduction, as the replicated finding. The RL increment is positive across four independent runs (+9.4, +3.1, +4.2, +4.2 points), and the SFT level replicates across retrains (76.0% and 77.1% at n=192; 68.8% at n=64). The surviving failures are design errors the verifier pinpoints (disconnected graphs, too few components), not formatting.

Re-measured at HEAD and at 7B. Everything above was graded by rubric v1. Rented A100s let us measure what that costs. We re-ran the recipe end to end at HEAD under rubric v3, which carries the width-propagation and orphan checks v2 added (Section 5) plus the vacuity fix of Section 3, with data regenerated under the same holdout exclusion (2,824 verified pairs survive from a nominal 3,000 draw at seed 20260716; 176 collided with the holdout and were dropped). The lift survives the stricter grader and the level drops, which is the direction a stronger blocker set can move a pass rate and the direction Section 11 predicted: Qwen2.5-1.5B goes from 15.1% (29/192, [11, 21]) to 69.8% (134/192, [63, 76]) after SFT, against 17.2% to 76.0% under v1. A 55-point lift with non-overlapping intervals is what the training data is worth once the verifier stops flattering the model.

Scaling the policy five-fold reverses the starting order and confirms the recipe. Qwen2.5-7B-Instruct *starts below* the 1.5B, 4.7% (9/192, [2, 9]) against 15.1%, because the larger instruct model is more verbose and fails to emit a parseable action plan on 84 of 192 tasks against the smaller model's 72. It *ends above*, at 76.0% (146/192, [70, 82]), a 71-point lift, with parse failures down to 15. Untrained pass@1 in this domain therefore measures format compliance rather than design skill, and the ordering by capability appears only after the environment has supplied the format. Transfer to the twelve hand-authored curated tasks moves in the same order (2/12 → 5/12 for the 7B, 2/12 → 4/12 for the 1.5B, n=12, suggestive). Both curated numbers are measured under the fixed task set (Section 3), so no free point is included.

The RL stage of this re-measurement is reported for the 1.5B policy only. At 7B, GRPO's log-probability pass over a 152k-token vocabulary at 1,408 sequence positions exhausts an 80GB card at group size 4. We did not shrink the group to fit, because the group size is part of the recipe. The v1-protocol RL results above stand as published. What the re-measurement adds is that the supervised half of the recipe holds at HEAD and at 5× the policy size.

A contamination audit. These are the *corrected* numbers. Our own contamination audit found that the original sampler drew parameters from small discrete lists, which yielded only 127 distinct tasks. Despite disjoint seeds, every task in the original eval split had appeared in training. The naive protocol measured 85.9% ([75, 92]) where the same-split number without overlap is 68.8% (44/64, [57, 79]). We widened the sampler to broad ranges (17,462 distinct tasks in a 20k draw at seed 999, reproducible without keys and validated by the released satisfiability sweep), added an explicit holdout exclusion to the data generator (re-verified: 0 of the 192 evaluation tasks appear among the 3,010 training rows), and re-ran the pipeline. The 17-point gap between protocols is itself a measurement, the memorization share [19] of the naive number, and the audit script ships with the environment.

Scaling, replication, and transfer. Data scaling is steep and non-monotonic at the low end (Figure 6; corrected zero-overlap protocol, n=192). 236 and 474 verified pairs *underperform* the raw instruct model (12.5% and 15.1% vs. 17.2%): a half-learned edit format displaces the base model's occasional valid outputs before replacing them. 945 pairs cross above the baseline (26.6%) and 3,010 reach 76.0%. Parse failures fall monotonically with data (97 → 58 → 56 → 14 of 192). The same U-shape appeared under the pre-audit protocol (9.4%, 14.1%, 34.4% against a 25.0% baseline), so the non-monotonic onset belongs to the recipe rather than to one split. At any single point the dip is within Wilson-interval overlap. What replicates across the two protocols is the shape: dip, crossover near 1k pairs, steep rise. Transfer tracks data diversity, and the audit gave us a controlled comparison. Models fine-tuned on the v1 data (127 distinct tasks) left the twelve hand-authored curated tasks unchanged (3/12, the untrained level), while the same recipe on the diverse v2 data doubles them to 6/12 (n=12, suggestive; surviving failures are substantive, e.g. missing attention). Diversity within the ten families is the lever for the curated split, and the generator makes it a parameter rather than a labeling effort. It does not carry to families the model never saw, which the next paragraph measures.

Held-out families: transfer is negative. The comparison above is n=12 and stays inside the ten families the generator produces, so it bounds transfer weakly. The generator makes the sharp test free, and we ran it: fine-tune on six families (`mlp`, `ae`, `cnn`, `fix`, `trim`, `grow`) and evaluate on the four the model has never seen (`txf`, `gqa`, `tower`, `norm`), same seed-999 split, rubric v3, 6,000 pairs. Within distribution the recipe behaves as everywhere else in this section: $44/308 \rightarrow 243/308$ ($14.3\% \rightarrow 78.9\%$). Out of distribution it does worse than fail to transfer. It destroys what the base model had: $46/204 \rightarrow \mathbf{0/204}$ ($22.5\% \rightarrow 0\%$, mean reward $0.36 \rightarrow -0.09$). The failure is structural rather than stylistic. 88 of the 204 held-out attempts return a graph whose input does not reach its output and 56 do not parse, so the fine-tuned policy has learned an edit pattern that is well-formed only for the shapes it was shown. The held-out families are also the ones the *untrained* model handled better (22.5% against 14.3%), so a difficulty artifact cannot explain the collapse. At this scale the environment teaches a per-family edit style, not a general design skill. A generator that produces only what it already covers does not fill a gap outside its coverage, and the scope of the training-data claim in this section is transfer *within* the generated distribution. We report it because it is the measurement most likely to be run against us.

RL alone from the raw model does not work. The released GRPO [6] loop runs end to end against the reward server on the same hardware, but RL *from the raw instruct model* does not move held-out metrics. Two runs (learning rates $10^{-6} \times 150$ and $10^{-5} \times 250$ steps, every checkpoint evaluated and reported) sit at or below their own baselines. The SFT result explains why: the raw policy cannot emit parseable edits on a third of tasks, so the group-relative policy gradient has no signal. This is the standard SFT-then-RL split. Supervised learning teaches the action format and RL then refines it, and the environment supplies both stages: the verified SFT data and the deterministic reward. Completing the recipe confirms it. Under the corrected protocol, 100 GRPO steps from the (merged) SFT checkpoint lift strictly unseen pass@1 from 76.0% to $\mathbf{80.2}\%$ (n=192), and an independent GRPO rerun from the same SFT checkpoint reproduces it exactly ($154/192$ both times). Mean reward rises $1.06 \rightarrow 1.12$ (1.10 on the rerun). The parse-failure reduction ($14 \rightarrow 10$) does not replicate (14 on the rerun), and we do not claim it. Re-run at HEAD under rubric v3 the stage lands in the same place from a lower start: $69.8\% \rightarrow \mathbf{79.7}\%$ ($134/192 \rightarrow 153/192$, $[73, 85]$), parse failures $13 \rightarrow 11$, mean reward $0.96 \rightarrow 1.11$. The stricter rubric costs the SFT checkpoint six points and the RL stage gives them back, so the recipe's endpoint is unchanged and the gain attributed to RL is larger under the grader that propagates shapes. The pre-audit chain showed the same pattern ($79.7 \rightarrow 89.1\%$). The same loop that did nothing on the raw policy works once the policy can act, and every surviving failure is a design error (disconnected two-tower graphs, component-count misses), not formatting.

Learning to consume verifier feedback. The amplification result (Section 6) shows that frontier models consume the verifier's failure messages *zero-shot*: one repair round closes their gap. A 1.5B model does not. Prompted with a failing attempt and the verifier's exact report, it repairs only 3% of edit-in-place tasks, because it has never seen a feedback-shaped prompt (a format shift the SFT above did not cover). The environment supplies the fix for free. We generate 204 repair-conditioned supervised pairs (a corrupted attempt plus the verifier's failure text as input, the reference edit as target, every corruption re-graded to confirm it fails) and fine-tune on them alongside the base data. Repair pass@1 rises from 3% to 52.6% ($80/152$), and 100 GRPO steps on top lift it to $\mathbf{69.1}\%$ ($105/152$, parse failures $\rightarrow 0$, mean reward $0.82 \rightarrow 1.04$), while strictly held-out plain design skill is unchanged (78% before and after). The same SFT-then-RL split that teaches the action format teaches feedback consumption one level up, and the teaching data is again produced by the environment at zero labelling cost. This repair split shares the training distribution. It shows that feedback consumption is teachable and RL-refinable, not out-of-distribution transfer.

G REWARD-MODEL AUDIT, IN FULL

Near-miss provenance, row by row. The near-miss tier is n=200 per model at seed 11, except `qwen-2.5-72b-instruct` at n=182 after 18 provider errors. The corruption mix is 58% dropped component, 33% dropped connection and 9% broken divisibility, each re-graded to confirm it fails. The llama rows are a re-run through OpenRouter at HEAD (blatant n=60). The grok row is a re-run at HEAD with the served id read from the response: an earlier draft carried 26.0% under the requested id `grok-4`, and grok-4.3, which is what that id resolves to, reads 24.5% over the same

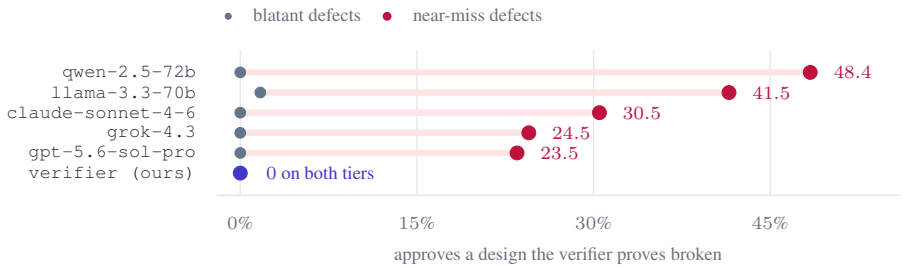


Figure 9: The collapse, visually. Ten of the eleven audited reward models score 0% false-positive on blatantly broken designs (Table 4). On near-miss negatives, passing designs one edit from correct, the five re-audited across five labs approve 23.5 to 48.4% (n=200 each). `llama-3.3-70b` is the one judge whose blatant rate is not zero at HEAD (1.7%). The model that was strongest on blatant defects is the weakest here. The verifier is 0% on both tiers by construction.

Failure category (curated split, `claude-sonnet-4-6`)	count
missing required layer type (e.g. attention)	3
over parameter budget	1
passed	**8 / 12**

Table 9: First-try failure reasons of `claude-sonnet-4-6` on the curated production-architecture split under rubric v2. 4 of 12 tasks fail, and every reason is detected deterministically. An independent rerun of the same split landed at 7/12, so treat the curated split as a 7 to 8 of 12 measurement: it is twelve tasks, and one task is inside run-to-run variance. On the same split `grok-4` passes 9/12, and all three of its failures carry a shape-mismatch blocker.

200 examples with zero provider errors. The two intervals overlap almost entirely, so the figure replicates. What changed is that we can now say which model produced it. The `gpt-5.6-sol-pro` row is transcribed from a logged run whose record we did not keep, which is why its false-negative cell is empty and why it is the one row in that table without a committed artifact. This tier is 3.3× the sample of the n=60 measurement in an earlier draft and replicates it (46.7 → 48.4, 33.3 → 30.5, 25.5 → 26.0 for the grok row before its re-run), with the ordering unchanged and every interval overlapping.

Model Identifier Resolution and Shared Blind Spots. The four Grok rows were re-measured on 2026-09-02 with the served model id read back from each response. `grok-4` and `grok-4-fast`, which an earlier version of Table 4 listed as two models, are both retired ids that xAI answers with grok-4.3, so that table counted one model twice. The harness had been recording the id it requested rather than the one that replied, and now records both. The per-example records settle a claim the earlier artifacts could not support: all four models fail on the *same three* examples (4, 24, 44 of the seed-7 set), and the two that miss a fourth each miss a different one, so the blind spot belongs to the family rather than to a checkpoint. A repeat run of `grok-4.20` moved between 57/60 and 56/60, which is the run-to-run variance these rows should be read against.

Accuracy Collapse of Reward Models on Near-Miss Defects. We re-run the audit with *near-miss* negatives: passing designs corrupted by a single edit (a dropped connection, an attention head count bumped off divisibility, a dropped component), each re-graded to confirm it fails. On these, the same judges collapse (Table 3, Figure 9). Claude approves 30.5% of provably broken designs (Wilson [25, 37], n=200), Grok 24.5% [19, 31], and Qwen-2.5-72B, the strongest of the three on blatant negatives (99.5% agreement, a single error in 200), approves 48.4% [41, 56] while rejecting almost nothing. It approves whatever looks plausible. The inversion is the point: LLM reward models are trustworthy when errors are obvious, and fail where reward hacking [20] happens, on subtle defects one edit from correct. The deterministic verifier is 0% on both difficulty tiers by construction. This is a use of the environment beyond training or evaluation: a ground-truth anchor that shows when the LLM reward models labs deploy can and cannot be trusted.

H COMPARISON WITH RELATED ENVIRONMENTS

Selecting among unexecuted candidates. Two concurrent lines measure the decision a verifier in an agent loop is asked to inform: which of several proposed candidates deserves execution. Zheng et al. [25] frame it as pairwise preference over agent-written ML solutions and reach 61.5% with a frontier model reading the code. Foster et al. [26] place a research preference model at the child-creation step of a search agent, reach up to 69.4% pairwise, and recover most of a validation oracle's end-to-end gain. LLMRouter [27] gives the general routing problem a five-part formulation (context encoder, candidate encoder, scoring function, decision rule and learning signal) together with a replayable benchmark in which every candidate was executed once and any router is scored offline against the record. In that vocabulary the two preference models and our verifier share the learning signal, executed outcomes, and differ in the scoring function (a language model versus a structural check) and in the decision rule: theirs is forced choice, ours abstains where the score ties, and Section 5 measures what that abstention costs. We adopt the replay design for our own selection measurements, so that a frontier judge, a size heuristic and the static score are compared on identical trained outcomes.

I VERIFIED REASONING TRACES

The same verifier also filters the input that reasoning-model post-training consumes: (spec $\rightarrow$ reasoning $\rightarrow$ design) triples where the design is re-graded and only passing traces are kept (rejection sampling). No trace enters the set unless its design solves the spec. A frontier model's verified yield under this filter, the fraction of its reasoned designs that pass, was 72.5% (327 of 451 tasks, with 49 API failures excluded) on a 500-task run. A second model (grok-4, on the diverse v2 generator) yields 87.8% and 87.4% across two independent 500-task runs, with zero API failures in either. The released tool records the yield per run and scales to larger sets. The released sets are additionally filtered by rubric v2, whose stricter width check (Section 5) removes the width-blind-spot class, leaving 306 claude traces and, for grok, a 392-trace union of the two runs. Every released row passes both rubric versions. Both sets are released publicly (links withheld for double-blind review). Because the split is seed-addressable, a private seed produces reasoning data whose designs never appeared on the public web.

J LIMITATIONS, IN FULL

The grounding study and the oracle row are measured under rubric v2. The supervised half of the training chain has since been re-run at HEAD under rubric v3 (Section 8), where it lost roughly six points and kept its lift. The calibration, amplification and reward-model measurements have also been re-run at HEAD, and levels moved by a few points in either direction without changing the effects (Sections 4, 6). The RL stage, the last to carry v1 numbers, has been re-run under rubric v3 as well: $69.8\% \rightarrow 79.7\%$, against $76.0\% \rightarrow 80.2\%$ under v1. No measurement in this paper now rests on the superseded rubric. The health-score magnitude is not a quality ranking (Section 5). The reward path does not execute PyTorch per rollout, so the verifier guarantees that the checks it implements hold exactly, not that no defect outside them exists. We widened the grounding study from three corruption families to seven and report what the four new ones expose. Generated references lean on `replace_model` for design-from-spec families, which biases imitation style. Each of these is documented with its measurement or its planned mitigation in `ROBUSTNESS.md`. A second caveat of the same kind: the harness recorded the model id it *requested*, not the one the provider served, so a retired id that resolves silently, as `grok-4` does to grok-4.3, would be credited to a model that never saw the prompt. Every provider path now reads the served id back from the response, the reward-model audit re-ran under it (Section 9), and Grok rows measured before that fix name what we asked for rather than what answered. Anthropic and OpenRouter return the id requested; xAI does not. One more caveat belongs here rather than in a footnote. The harness's OpenAI-compatible provider path sent no token budget, so a reasoning model reached through it could spend the provider's default budget on reasoning tokens and return an empty completion, which every caller downstream scored as a failed design. This deflated pass rates and inflated judge false-negative rates on that path without raising an error. It is fixed, an empty completion now raises, and every number in this paper taken through that path has been re-measured. Any earlier artifact

routed that way should be read with this in mind. The training experiments use two small policies (Qwen2.5-1.5B and 7B, LoRA) with evaluation splits of n=192 (headline) and n=64 (replications). The larger policy is measured through supervised fine-tuning only, because its RL stage does not fit one 80GB card at the published group size. The SFT and RL gains are in-distribution, and we measured how sharply: held-out families collapse from an untrained 22.5% to 0% while seen families reach 78.9% (Section 8), so the recipe teaches a per-family edit style, and the curated-split signal ($2/12 \rightarrow 5/12$ at 7B) should be read in that light. Contamination against *model pretraining* is now measured (Section 3), but over one corpus and one index rather than every corpus a frontier model has seen. The amplification study, the near-miss audit, and the eleven-model blatant-tier audit were all re-run on generator v2 (diverse instances) with results matching v1, which resolves the earlier effective-sample-size concern for those measurements. Scaling the policy and diversifying the generated data further are the natural extensions.

K EXAMPLE CURATED TASKS

The twelve curated tasks are hand-authored from real production architectures. Figure 10 sketches four, as the typed graphs the environment operates on; each is a distinct family (vision, ranking, sequence, retrieval) with its own required layer types and budgets.

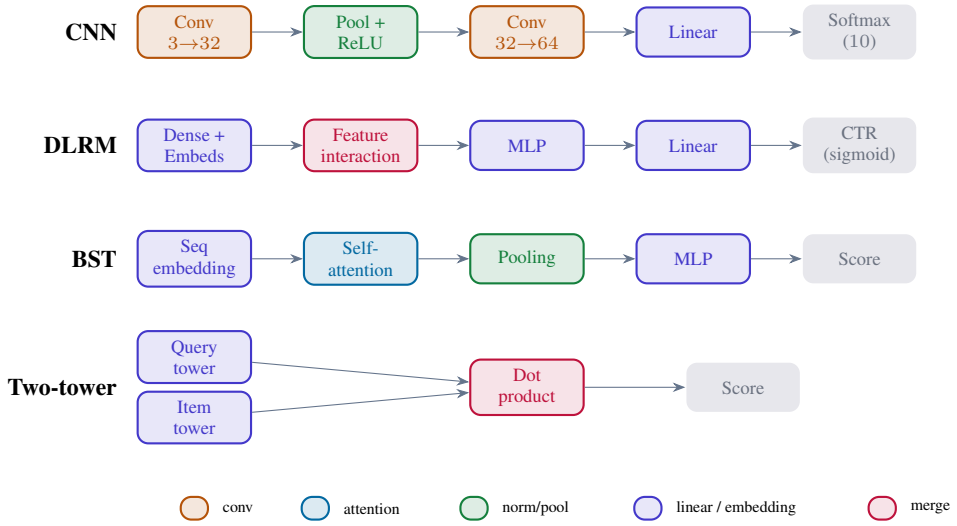


Figure 10: Four of the twelve curated tasks, as typed graphs, coloured by layer type (the same convention the graph editor uses): a CIFAR-10 CNN, a DLRM click-through-rate ranker, a Behavior Sequence Transformer (which *requires* an attention layer), and a two-tower retrieval model (two encoders scored by a dot product). These are the states an agent edits and the verifier grades.